6用户层、缓冲、调度

2018年12月24日 9:10

-
- 用户层的IO软件

一. 系统调用与库函数

1. 系统调用：OS在用户层引进的中介过程，应用程序可通过它间接调用IO过程
   1) OS捕获到系统调用后，将CPU切换到核心态，转换到相应过程，完成IO，再切换回用户态，继续运行应用程序
   2) 早期，系统调用是应用程序取得OS服务的唯一途径
   3) 早期的系统调用以汇编语言提供，只有汇编语言的程序可以调用
2. 库函数：高级语言和新操作系统如C语和UNIX中，系统调用一一对应的库函数
   1) 库函数与调用程序连接在一起，嵌入在运行时装入内存的二进制程序中
   2) 内核提供OS基本功能，库函数扩展了OS内核，使用户方便取得服务
   3) 微软也定义了一套Application Program Interface，不过与系统调用并不一一对应
   4) 许多系统调用本身就采用C语言编写，以函数形式提供，可在C语言编写的程序直接调用
   5)

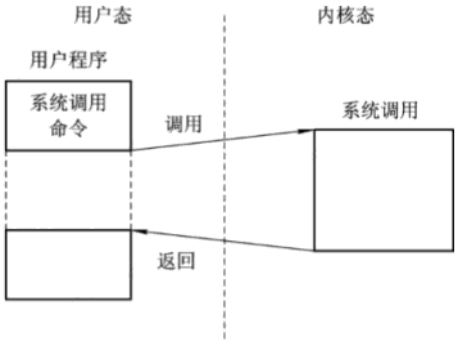
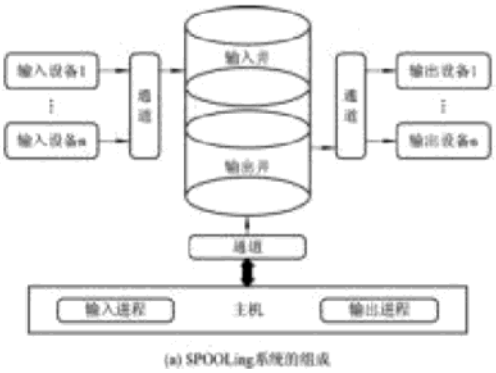


图6-20 系统调用的执行过程

二. 假脱机Spooling系统

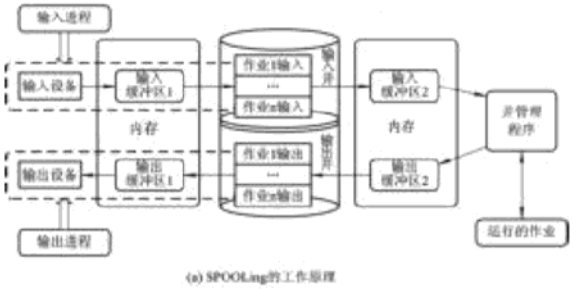
1) 20世纪50年代的脱机IO技术：用专门的外围控制机在低速IO设备和高速磁盘间传送数据

1. 多道程序系统中，可用两道程序模拟外围控制机的输入、输出，将一台物理IO设备虚拟为多台逻辑IO设备，把这种在联机情况下实现的**同时外围**操作称为SPOOLing(Simultaneaus Periphernal Operating On Line)，或称为假脱机
2. SPOOLing的组成
   1)

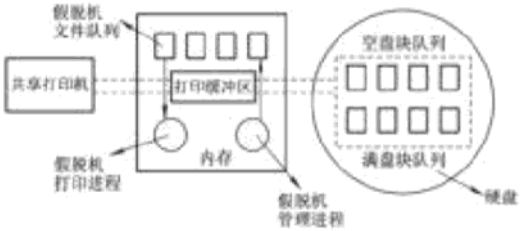


(a) SPOOLing系统的组成

   2) 输入/出井：**磁盘上的两个区域**，专门用于模拟脱机时的磁盘。用于以文件队列形式收容输入/出数据，称这些文件为井文件
   3) 输入/出缓冲区：内存中的两个缓冲区，用于缓和cpu和磁盘速度不匹配的矛盾

4) 预输入/缓输出进程：模拟外围控制机，（借助缓冲区）在设备和井间传数据
5) 井管理程序：控制作业与井之间的信息交换，作业提出请求时，由操作系统调用井管理程序，（借助另一个缓冲区）控制井内信息IO
6) 

(a) SPOOLing的工作原理

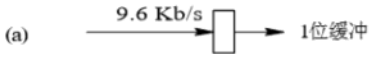
3. SPOOLing的特点
   1) 提高IO速度，**从操作低速IO设备变为操作磁盘**
   2) 将**独占设备改造成共享设备**，因为并没有把设备分配给进程，但磁盘中的空闲盘块和IO请求表是共享的
   3) 实现**虚拟设备**，每个进程都**以为自己独占了设备**
4. 假脱机打印机系统
   1) 打印机是常见独占设备，共享打印广泛用于多用户系统和局域网
   2) 

(c) 假脱机打印机系统的组成
   3) 假脱机管理进程：在磁盘缓冲区申请空闲盘块，暂存打印数据、申请打印请求表，填入打印要求，挂入井文件队列
   4) 假脱机打印进程：当打印机空闲，按井文件队首的打印请求表，将输出井的数据传送到内存缓冲区，交给打印机打印。若无打印请求就自我阻塞
   5) 打印操作是将数据送进缓冲区，在打印机空闲且排到队首时，cpu才用一个时间片进行打印。这些过程用户不可见
5. 守护进程daemon
   1) 用于执行一部分假脱机管理程序的功能，如申请空闲盘块，送入打印数据，返回盘块首址给进程，接收进程提交的完整打印请求等
   2) 是唯一允许使用守护的独占设备的进程，其他进程只能写请求文件
   3) 平常睡眠，出现新的井文件时被唤醒
   4) 除打印机外，服务器网络等也各可配置一个守护进程和井文件队列

- ◆
- ◆ 缓冲区管理

一. 缓冲的引入

1. 缓和 CPU 与 I/O 设备间速度不匹配的矛盾
2. 提高 CPU 和 I/O 设备之间的并行性
3. 减少对 CPU 的中断频率，放宽 CPU 中断响应时间的限制
   1) 如设置一个8位缓冲移位寄存器，即可将cpu中断响应时间放宽到8倍



2)

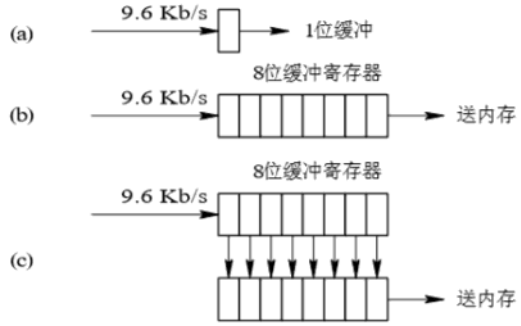
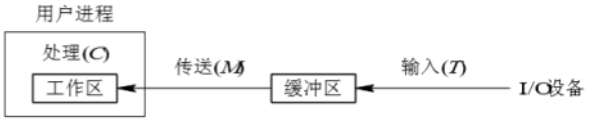


图 5-10　利用缓冲寄存器实现缓冲

4. 解决数据粒度不匹配问题
    1) 数据粒度：数据单元大小。越细化，粒度越小
    2) 粒度小的数据可以存一定量再操作，粒度大的数据可以分几次操作

二. 单缓冲区和双缓冲区

1. 单缓冲(Single Buffer)：进程每发出一 I/O 请求，系统便在主存分配一缓冲区
    1) 块设备输入时，假定从磁盘把一块数据输入到缓冲区的时间 为 T，数据传送到用户区的时间为 M，而 CPU 对这一块数据处理(计算)的时间为 C。由于 T 和 C 可并行，当 T>C 时，系统对每一块数据的处理时间为 M+T，反之则为 M+C，故可把系统对每一块数据的处理时间表示为 Max(C，T)+M



2)

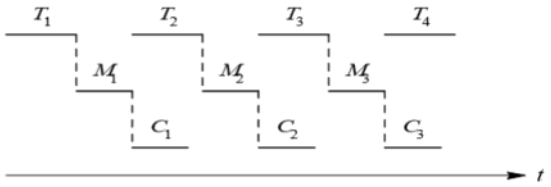
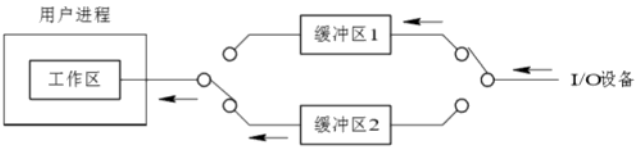


图 5-11　单缓冲工作示意图

    3) 新一行数据输入时进程被挂起；旧一行数据输出时进程被阻塞
2. 双缓冲(Double Buffer) /缓冲对换(Buffer Swapping)
    1) 第一缓冲区数据传送时，不用等待，转而使用第二缓冲区即可
    2) 处理一块数据的时间为 Max{C+M，T}。如果 C+M<T，可使块设备连续输入；如果 C+M>T，则可使 CPU 不必等待设备输入
    3) 处理n块数据的时间为 T+Max{C+M，T}+C



4)

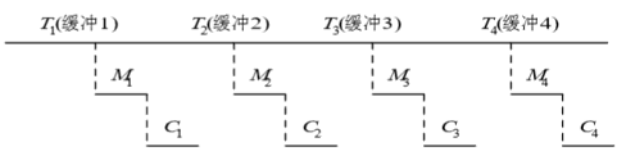


图 5-12　双缓冲工作示意图

    5) 双向数据传输同理，可以双方及其各设两个缓冲区

6)

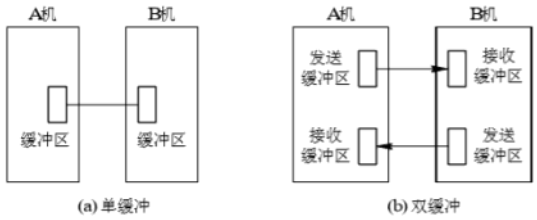


图 5-13　双机通信时缓冲区的设置

三. 环形缓冲区

1) 输入输出时间相差较大时双缓冲仍不够理想，需要多缓冲

1. 环形缓冲区的组成

1) 等大的多个缓冲区：空缓冲区 R、装满数据的缓冲区 G、计算进程正在使用的现行工作缓冲区 C

2) 多个指针：计算进程下个可用缓冲区 G 的指针 Nextg、输入进程下次可用的空缓冲区 R 的指针 Nexti、计算进程正在使用的缓冲区 C 的指针 Current

3)

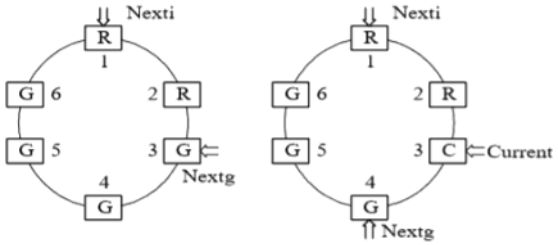


图 5-14　循环缓冲

2. 环形缓冲区的使用

1) Getbuf过程：

(1) 计算：将 Nextg 所指示的 R 提供给进程使用，改为C，并令 Current 指针指向其第一个单元，同时将 Nextg 移向下一个 G

(2) 输入：将指针 Nexti 所指示的 R 提供给输入进程，同时将 Nexti 指针移向下一个 R

2) Releasebuf过程：

(1) 计算：提取完毕 C 时调用，将 C 释放，改为 R

(2) 输入：装满 R 时调用，将 R 释放，改为 G

3. 进程间的同步问题

1) Nexti 指针追赶上 Nextg 指针：系统受计算限制

(1) 即输入过快，无空区，应阻塞输入进程直至计算进程调用 Releasebuf 过程

2) Nextg 指针追赶上 Nexti 指针：系统受 I/O 限制

(1) 即计算过快，无满区，应阻塞计算进程直至输入进程调用Releasebuf过程

四. 缓冲池Buffer Pool

1) 是即可输入又可输出的公用缓冲池

2) 缓冲区仅是内存块链表，缓冲池是数据结构+操作函数

3) 缓冲池的每个缓存区都有首部和缓冲体两部分

(1) 首部包含缓冲区号、设备号、数据块号、同步信号量、队列指针

1. 缓冲池的组成：空缓冲区；装满输入数据的缓冲区；装满输出数据的缓冲区；用于收容/提取 输入/输出数据的工作缓冲区

1) 空缓冲队列 emq。其队首指针 F(emq)和队尾指针 L(emq)分别指向其首缓冲区和尾缓冲区

2) 输入队列 inq。其队首指针 F(inq)和队尾指针 L(inq)分别指向其首缓冲区和尾缓冲区

3) 输出队列 outq。其队首指针 F(outq)和队尾指针 L(outq)分别指向其首缓冲区和尾缓冲区

2. Getbuf 过程和 Putbuf 过程

1) Addbuf(type,number)过程。将number所指示的缓冲区B挂在type 队列上
2) Takebuf(type)过程。从 type 所指示的队列的队首摘下一个缓冲区
3) 缓冲池中的队列本身是临界资源，访问队列，既应互斥，又须同步，可为每队列设置一个互斥信号量MS(type)。 此外，为了保证诸进程同步地使用缓冲区，又为每个缓冲队列设置了一个资源信号量RS(type)

(1) Procedure Getbuf(type){
Wait(RS(type));
Wait(MS(type));
B(number)=Takebuf(type);
Signal(MS(type));}

(2) Procedure Putbuf(type,number){
Wait(MS(type));
Addbuf(type,number);
Signal(MS(type));
Signal(RS(type));}

3. 缓存区工作方式

1)

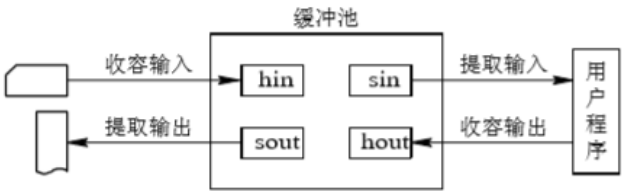


图 5-15 缓冲区的工作方式

2) 收容输入。输入进程输入数据时，调用 Getbuf(emq)过程，从空缓冲队列 emq 队首摘下一空缓冲区，作为收容输入工作缓冲区 hin。输入数据， 再调用 Putbuf(inq，hin)过程，将该缓冲区挂在输入队列 inq 上
3) 提取输入。计算进程需要数据时，调用 Getbuf(inq)过程，从输入队列 inq 队首取得一个缓冲区，作为提取输入工作缓冲区 sin 。计算进程用完该数据后，再调用 Putbuf(emq，sin)过程，将该缓冲区挂到空缓冲队列 emq 上
4) 收容输出。计算进程需要输出时，调用 Getbuf(emq)过程从空缓冲队列 emq 队首取得一空缓冲区，作为收容输出工作缓冲区 hout。装满输出数据后，又调用 Putbuf(outq，hout)过程，将该缓冲区挂在 outq 末尾
5) 提取输出。由输出进程调用 Getbuf(outq)过程，从输出队列的队首取得一装满输出数据的缓冲区，作为提取输出工作缓冲区 sout。在数据提取完后，再调用 Putbuf(emq，sout) 过程，将该缓冲区挂在空缓冲队列末尾

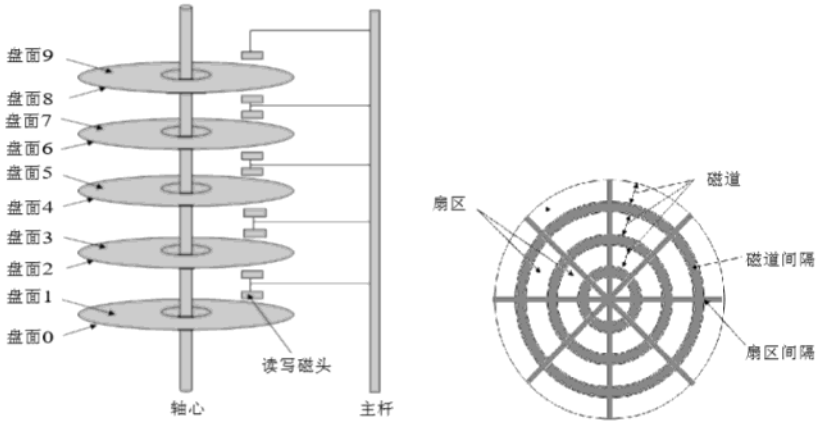
-
- 磁盘存储器的性能和调度

1. 提高磁盘性能除了考改善寻道调度算法外还可靠提高IO速度、冗余技术

一. 磁盘性能简述

1. 数据的组织和格式
   1) 磁盘设备包括一或多个物理盘片，每片分一或两个存储面(surface)
   2) 每面被组织成若干同心环，称为柱面/磁道(track)，各磁道间留有间隙gap
   3) 每条磁道上可存储同数目的二进制位，内层磁道的磁盘位数密度更高
   4) 每条磁道逻辑上划分成若干个扇区(sectors)，软盘 8 ~ 32 个，硬盘数百
   5) **扇区又称盘块**(或数据块)。各扇区间保留一定扇区间隙gap

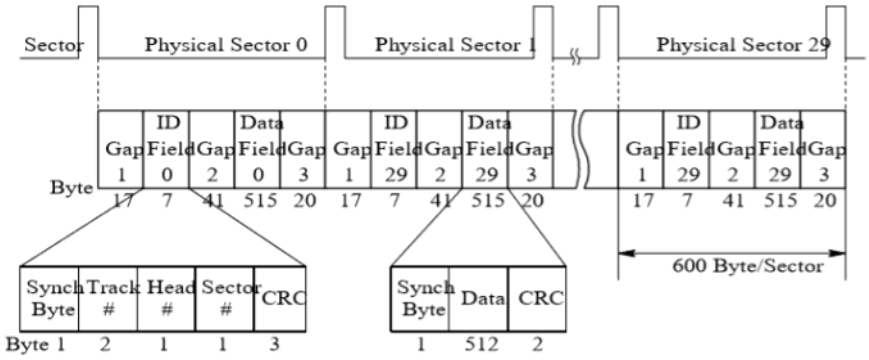
(1)



(a) 磁盘驱动器的结构　　(b) 磁盘的数据布局

6) 现代磁盘常把盘面划分成若干环带，外层环带拥有更多扇区
7) 大多数磁盘都隐藏了细节，只向系统提供虚拟几何的磁盘规格
8) 磁盘存储数据前需要先低级格式化，如温切斯特盘，每磁道30扇区
   (1) 标识符字段：每个扇区的其中88字节。其中一个字节的 SYNCH 具有特定的位图像，作为该字段的定界符， 利用磁道号、 磁头号及扇区号三者来标识一个扇区；CRC 字段用于段校验
   (2) 数据字段：每个扇区的剩余 512 个字节，用于存储数据

(3)



9) 磁盘低级格式化后即可分区，每个分区是逻辑上独立的盘，每个分区的起始扇区都记录在磁盘0扇区的分区表，该表中有一个被标记成活动的主引导记录分区，保证从硬盘能引导系统
10) 真正使用磁盘前还需再高级格式化一次，设置引导块、空闲存储管理、根目录、空文件系统，再在分区表标记其文件系统

2. 磁盘的类型
   1) 硬盘、软盘；单片盘、多片盘；固定头、活动头/移动头磁盘
   2) 固定头磁盘：每条磁道上都有一读/写磁头，所有的磁头都被装在一刚性磁臂中，有效地提高了磁盘的 I/O 速度。主要用于大容量磁盘上
   3) 移动头磁盘：每个盘面仅配有一个磁头，也被装入磁臂中。为能访问该盘面上的所有磁道，该磁头必须能移动寻道，仅能以串行方式读/写，致使 I/O 速度较慢； 由于其结构简单，仍广泛应用于中小型磁盘设备中。在微型机上配置的温盘和软盘都采用移动磁头结构
3. 磁盘访问时间
   1) 寻道时间 Ts： 磁臂(磁头)移动到指定磁道上所经历的时间
      (1) 是启动磁臂的时间 s 与磁头移动 n 条磁道所花费的时间之和
         i. $T_s = m \times n + s$
         ii. m与磁盘驱动器速度有关，一般磁盘m=0.2；高速磁盘m<=0.1
         iii. s=2ms，一般温盘的寻道时间约5~30ms
   2) 旋转延迟时间 Tr： 指定扇区移动到磁头下面所经历的时间

(1) 不同的磁盘类型中，旋转速度至少相差一个数量级，如软盘为300 r/min，硬盘一般为7200～15 000 r/min，甚至更高

(2) 硬盘，每转需时4 ms，平均旋转延迟时间Tr为2 ms；而软盘平均Tr为50～100 ms

3) 传输时间Tt：把数据从磁盘读出或向磁盘写入数据所经历的时间

(1) 与读/写字节数b和旋转速度有关

(2) $$T_t = \frac{b}{rN}$$

(3) r为磁盘每秒钟的转数；N为一条磁道上的字节数

4) 当一次读/写的字节数相当于半条磁道上的字节数时，Tt与Tr相同，此时

(1) $$T_a = T_s + \frac{1}{2r} + \frac{b}{rN}$$

(2) 寻道时间和旋转延迟时间基本上与所读/写数据多少无关，且通常占据了访问时间中的大头

(3) 现传输速率已高达80m/s，提高传输效率最快的方式已是适当地集中数据传输了

二. 早期的磁盘调度算法

1. 先来先服务(First Come First Served)

1) 公平、简单、不会使某进程长期得不到满足；平均寻道时间长

2) 平均寻道距离大，只适于磁盘IO进程数少的场合

2. **最短寻道时间优先**(Shortest Seek Time First)

1) 即选择最近的磁道。仍不能保证平均寻道时间最短，仅优于fcfs

(从100号磁道开始)	
被访问的下一个磁道号	移动距离(磁道数)
55	45
58	3
39	19
18	21
90	72
160	70
150	10
38	112
184	146
平均寻道长度: 55.3	

图 5-25 FCFS 调度算法

(从100号磁道开始)	
被访问的下一个磁道号	移动距离(磁道数)
90	10
58	32
55	3
39	16
38	1
18	20
150	132
160	10
184	24
平均寻道长度: 27.5	

图 5-26 SSTF 调度算法

三. 基于扫描的磁盘调度算法

1) 进程“饥饿”(Starvation)现象：新请求不断到达，低优先级进程永不被满足

1. **扫描**(SCAN)算法/电梯调度算法

1) SCAN 算法优先考虑磁头当前移动方向，正向无磁道需访问时才反向移动

2. **循环扫描**(CSCAN)算法

1) 扫描算法不能照顾到磁头换向后新到的进程

2) 循环扫描使磁头移到最外道后马上回到最里

3) T为扫描一轮的时间，Smax为移动的时间，则新到进程等待时间由2T变为T+Smax

4)

(从100#磁道开始，向磁道号增加方向访问)	
被访问的下 一个磁道号	移动距离 (磁道数)
150	50
160	10
184	24
90	94
58	32
55	3
39	16
38	1
18	20
平均寻道长度：27.8	

图 5-27　SCAN 调度算法示例

(从100#磁道开始，向磁道号增加方向访问)	
被访问的下 一个磁道号	移动距离 (磁道数)
150	50
160	10
184	24
18	166
38	20
39	1
55	16
58	3
90	32
平均寻道长度: 35.8	

图 5-28　CSCAN 调度算法示例

3. NStepSCAN 和 FSCAN 调度算法
   1) NStepSCAN
      (1) “磁臂粘着” (Armstickiness)：进程反复请求对某磁道的I/O 操作，垄断了整个磁盘设备，磁臂停留在某处不动
      (2) N步扫描将磁盘请求队列分为若干长为N的子队列，FCFS处理
      (3) 通过把新进程的请求放在其他子队列，避免粘着
      (4) n=1时退化为fcfs，n很大时接近scan
   2) FSCAN
      (1) N步扫描的退化，只分成当前子队列和新到子队列
          i.
          ii.
          iii.
          iv.
          v.
          vi.
          vii.
          viii.
          ix. ---------我是底线-------